



Science and  
Technology  
Facilities Council



# GALAHAD

# MOP

USER DOCUMENTATION

GALAHAD Optimization Library version 5.1

## 1 SUMMARY

GALAHAD\_MOP is a suite of Fortran 90 procedures for **performing operations on/with a matrix  $\mathbf{A}$**  of derived data type `SMT_type` (Section 2.3), which allows for multiple storage types (Section 2.1). In particular, this package contains the following subroutines for a given  $m$  by  $n$  matrix  $\mathbf{A}$ :

- subroutine `mop_Ax` computes matrix-vector products of the form

$$\mathbf{r} \leftarrow \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{r} \quad \text{and} \quad \mathbf{r} \leftarrow \alpha \mathbf{A}^T \mathbf{x} + \beta \mathbf{r}$$

for given scalars  $\alpha$  and  $\beta$ , and vectors  $\mathbf{x} \in \mathbb{R}^n$  and  $\mathbf{r} \in \mathbb{R}^m$ ;

- subroutine `mop_getval` obtains the  $(i, j)$ -element of the matrix  $\mathbf{A}$  for given integers  $i$  and  $j$ ;
- subroutine `mop_scaleA` scales the rows of  $\mathbf{A}$  by a given vector  $\mathbf{u} \in \mathbb{R}^m$  and the columns by a vector  $\mathbf{v} \in \mathbb{R}^n$ .

**ATTRIBUTES — Versions:** `GALAHAD_MOP_single`, `GALAHAD_MOP_double`, **Uses:** `GALAHAD_SMT_double`. **Date:** November 2009. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, and D. P. Robinson, University of Oxford, UK. **Language:** Fortran 90.

## 2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_MOP_single
```

with the obvious substitution `GALAHAD_MOP_double`, `GALAHAD_MOP_quadruple`, `GALAHAD_MOP_single_64`, `GALAHAD_MOP_double_64` and `GALAHAD_MOP_quadruple_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `SMT_type` (Section 2.3) and the subroutines `MOP_Ax`, `MOP_getval`, and `MOP_scaleA` (Sections 2.4, 2.5, 2.6) must be renamed on one of the `USE` statements.

### 2.1 Matrix storage formats

The matrix  $\mathbf{A}$  may be stored in a variety of input formats.

#### 2.1.1 Dense storage format

The matrix  $\mathbf{A}$  is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Component  $n * (i - 1) + j$  of the storage array `A%val` will hold the value  $a_{ij}$  for  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ . If  $\mathbf{A}$  is symmetric, only the lower triangular part (that is the part  $a_{ij}$  for  $1 \leq j \leq i \leq n$ ) should be stored. In this case the lower triangle will be stored by rows, that is component  $i * (i - 1) / 2 + j$  of the storage array `A%val` will hold the value  $a_{ij}$  (and, by symmetry,  $a_{ji}$ ) for  $1 \leq j \leq i \leq n$ .

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

### 2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the  $l$ -th entry of  $\mathbf{A}$ , its row index  $i$ , column index  $j$  and value  $a_{ij}$  are stored in the  $l$ -th components of the integer arrays `A%row`, `A%col` and real array `A%val`. The order is unimportant, but the total number of entries `A%ne` is also required. If  $\mathbf{A}$  is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

### 2.1.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row  $i$  appear directly before those in row  $i + 1$ . For the  $i$ -th row of  $\mathbf{A}$ , the  $i$ -th component of an integer array `A%ptr` holds the position of the first entry in this row, while `A%ptr(m + 1)` holds the total number of entries plus one. The column indices  $j$  and values  $a_{ij}$  of the entries in the  $i$ -th row are stored in components  $l = \text{A\%ptr}(i), \dots, \text{A\%ptr}(i + 1) - 1$  of the integer array `A%col`, and real array `A%val`, respectively. If  $\mathbf{A}$  is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

### 2.1.4 Sparse column-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in column  $j$  appear directly before those in column  $j + 1$ . For the  $j$ -th column of  $\mathbf{A}$ , the  $j$ -th component of the integer array `A%ptr` holds the position of the first entry in this column, while `A%ptr(n + 1)` holds the total number of entries plus one. The row indices  $i$  and values  $a_{ij}$  of the entries in the  $j$ -th column are stored in components  $l = \text{A\%ptr}(j), \dots, \text{A\%ptr}(j + 1) - 1$  of the integer array `A%row`, and real array `A%val`, respectively. If  $\mathbf{A}$  is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

### 2.1.5 Diagonal storage format

If  $\mathbf{A}$  is diagonal (i.e.,  $a_{ij} = 0$  for all  $1 \leq i \neq j \leq n$ ) only the diagonal entries  $a_{ii}$  for  $1 \leq i \leq n$  should be stored, and the first  $n$  components of the array `A%val` should be used for this purpose.

## 2.2 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

## 2.3 The derived data type for holding the matrix A

The matrix  $\mathbf{A}$  is stored using the derived data type `SMT_type` whose components are:

- `m` is a scalar component of type `INTEGER(ip_)`, that holds the number of rows in the matrix.
- `n` is a scalar component of type `INTEGER(ip_)`, that holds the number of columns in the matrix.
- `ne` is a scalar variable of type `INTEGER(ip_)`, that holds the number of matrix entries.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

`type` is a rank-one allocatable array of type default `CHARACTER`, that is used to indicate the matrix storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of `H%type` must contain the string `DENSE`. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of `H%type` must contain the string `COORDINATE`, for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of `H%type` must contain the string `SPARSE_BY_ROWS`, for the sparse column-wise storage scheme (see Section 2.1.4), the first seventeen components of `H%type` must contain the string `SPARSE_BY_COLUMNS`, and for the diagonal storage scheme (see Section 2.1.5), the first eight components of `H%type` must contain the string `DIAGONAL`.

For convenience, the procedure `SMT_put` may be used to allocate sufficient space and insert the required keyword into `H%type`. For example, if we wish to store **A** using the co-ordinate scheme, we may simply

```
CALL SMT_put( A%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package `SMT` for further details on the use of `SMT_put`.

`val` is a rank-one allocatable array of type `REAL(rp_)` and dimension at least `ne`, that holds the values of the entries. Each pair of off-diagonal entries  $a_{ij} = a_{ji}$  of a *symmetric* matrix **A** is represented as a single entry (see §2.1.1–2.1.5).

`row` is a rank-one allocatable array of type `INTEGER(ip_)`, and dimension at least `ne`, that may hold the row indices of the entries (see § 2.1.2 and 2.1.4).

`col` is a rank-one allocatable array of type `INTEGER(ip_)`, and dimension at least `ne`, that may hold the column indices of the entries (see §2.1.2–2.1.3).

`ptr` is a rank-one allocatable array of type `INTEGER(ip_)`. If sparse row-wise storage is used, then `ptr` should be of dimension at least `m + 1` and should contain pointers to the first entry in each row (see §2.1.3). If sparse column-wise storage is used, then `ptr` should be of dimension at least `n + 1` and should contain pointers to the first entry in each column (see §2.1.4).

## 2.4 The subroutine to form matrix-vector products

The subroutine `MOP_Ax` may be called to compute matrix vector products with **A** of the form

$$\mathbf{r} \leftarrow \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{r} \quad (2.1)$$

or

$$\mathbf{r} \leftarrow \alpha \mathbf{A}^T \mathbf{x} + \beta \mathbf{r} \quad (2.2)$$

by using

```
CALL MOP_Ax( alpha, A, X, beta, R, [out, error, print_level, symmetric, transpose] )
```

where square brackets indicate OPTIONAL arguments.

`alpha` is a scalar `INTENT(IN)` argument of type `REAL(rp_)` that must hold the value of  $\alpha$ .

`A` is a scalar `INTENT(IN)` argument of type `SMT_type` (see Section 2.3) that must hold the matrix **A**.

`X` is a rank-one `INTENT(IN)` array of type `REAL(rp_)` that must contain the components of the vector **x**.

`beta` is a scalar `INTENT(IN)` argument of type `REAL(rp_)` that must hold the value of  $\beta$ .

`R` is a rank-one `INTENT(INOUT)` array of type `REAL(rp_)` that must contain the components of the vector **r**. `R` need not be set on entry if `beta` is zero.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

`out` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for informational messages. If this argument is not provided, then the default value `out = 6` is used.

`error` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for error messages. If this argument is not provided, then the default value `error = 6` is used.

`print_level` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that is used to control the amount of informational output which is required. No informational output will occur if `print_level ≤ 0`. If `print_level = 1`, minimal output will be produced and if `print_level ≥ 2` then output will be increased to provide full details. The default is `print_level = 0`.

`symmetric` is an OPTIONAL scalar INTENT(IN) argument of type default LOGICAL that should be set `.TRUE.` if the matrix **A** is symmetric, and set `.FALSE.` otherwise. If this argument is not provided, then the default value of `.FALSE.` is used.

`transpose` is an OPTIONAL scalar INTENT(IN) argument of type default LOGICAL that should be set `.FALSE.` if the user wishes to compute (2.1), and set `.TRUE.` if the user wishes to compute (2.2). If `transpose` is not provided, then the default value of `.FALSE.` is used.

### 2.4.1 Warning and error messages

All warning and error messages will be printed on unit `error` as discussed in the previous section.

### 2.4.2 Information printed

If `print_level` is positive, information about the calculation will be printed on unit `out` as discussed previously. In particular, if `print_level = 1`, then the values `symmetric`, `transpose`, `A%m`, `A%n`, `A%type`, `A%id`, `alpha`, and `beta` are printed. If `print_level = 2`, then additionally `A%ptr`, `A%val`, `A%row`, and `A%col` are printed. If `print_level ≥ 3`, then additionally the input **X** and **R** as well as the result **R** will be printed.

## 2.5 The subroutine to get matrix values

The subroutine `MOP_getval` may be used to get the  $(i, j)$ -th element of the matrix **A** by using

```
CALL MOP_getval( A, row, col, val, [symmetric, out, error, print_level] )
```

where square brackets indicate OPTIONAL arguments.

`A` is a scalar INTENT(IN) argument of type `SMT_type` (see Section 2.3) that must contain the matrix **A**.

`row` is a scalar INTENT(IN) argument of type INTEGER(ip\_) that specifies the row index  $i$  of the requested element of the matrix **A**.

`col` is a scalar INTENT(IN) argument of type INTEGER(ip\_) that specifies the column index  $j$  of the requested element of the matrix **A**.

`val` is a scalar INTENT(OUT) argument of type REAL(rp\_) that holds the value of the  $(i, j)$ -th element of the matrix **A** on return.

`symmetric` is an OPTIONAL scalar INTENT(IN) argument of type default LOGICAL that should be set `.TRUE.` if the matrix **A** is symmetric, and set `.FALSE.` otherwise. If `symmetric` is not provided, then the default value of `.FALSE.` is used.

`out` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for informational messages. If this argument is not provided, then the default value `out = 6` is used.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

`error` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for error messages. If this argument is not provided, then the default value `error = 6` is used.

`print_level` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that is used to control the amount of informational output which is required. No informational output will occur if `print_level ≤ 0`. If `print_level = 1`, minimal output will be produced and if `print_level ≥ 2` then output will be increased to provide full details. The default is `print_level = 0`.

### 2.5.1 Warning and error messages

All warning and error messages will be printed on unit `error` as discussed in the previous section.

### 2.5.2 Information printed

If `print_level` is positive, information about the subroutine data will be printed on unit `out` as discussed previously. In particular, if `print_level = 1`, then the values `A%m`, `A%n`, `A%type`, `A%id`, `row`, `col`, and the resulting value `val` are printed. If `print_level ≥ 2`, then additionally `A%ptr`, `A%val`, `A%row`, and `A%col` are printed.

## 2.6 The subroutine to scale the matrix

The subroutine `MOP_scaleA` may be called to scale the rows of the  $m$  by  $n$  matrix  $\mathbf{A}$  by the vector  $\mathbf{u} \in \mathbb{R}^m$  and the columns by the vector  $\mathbf{v} \in \mathbb{R}^n$ . In other words, it forms the scaled matrix whose  $(i, j)$ -th element is  $u_i a_{i,j} v_j$ . This scaled matrix is stored in  $\mathbf{A}$  on return. If the OPTIONAL argument `symmetric` is set `.TRUE.`, then the rows and columns of  $\mathbf{A}$  are scaled by the vector  $\mathbf{u}$ . The calling sequence is given by

```
CALL MOP_scaleA( A, [u, v, out, error, print_level, symmetric] )
```

where square brackets indicate OPTIONAL arguments.

$\mathbf{A}$  is a scalar INTENT(INOUT) argument of type `SMT_type` (see Section 2.3) that must contain the matrix  $\mathbf{A}$ .

$\mathbf{u}$  is an OPTIONAL rank-one INTENT(IN) argument of type `REAL(rp_)` of length `A%m` whose  $i$ -th component is used to scale the  $i$ -th row of the matrix  $\mathbf{A}$ .

$\mathbf{v}$  is an OPTIONAL rank-one INTENT(IN) argument of type `REAL(rp_)` of length `A%n` whose  $i$ -th component is used to scale the  $i$ -th column of the matrix  $\mathbf{A}$ .

`out` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for informational messages. If this argument is not provided, then the default value `out = 6` is used.

`error` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that holds the stream number for error messages. If this argument is not provided, then the default value `error = 6` is used.

`print_level` is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip\_), that is used to control the amount of informational output which is required. No informational output will occur if `print_level ≤ 0`. If `print_level = 1`, minimal output will be produced and if `print_level ≥ 2` then output will be increased to provide full details. The default is `print_level = 0`.

`symmetric` is an OPTIONAL scalar INTENT(IN) argument of type default LOGICAL that should be set `.TRUE.` if the matrix  $\mathbf{A}$  is symmetric, and set `.FALSE.` otherwise. If `symmetric` is not provided, then the default value of `.FALSE.` is used.

### 2.6.1 Warning and error messages

All warning and error messages will be printed on unit `error` as discussed in the previous section.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

### 2.6.2 Information printed

If `print_level` is positive, information about the arguments will be printed on unit `out` as discussed previously. In particular, if `print_level`  $\geq 1$ , then the values `A%m`, `A%n`, `A%type`, `A%id`, `A%ptr`, `A%val`, `A%row`, `A%col`, `u` and `v` will be printed.

## 3 GENERAL INFORMATION

**Use of common:** None.

**Workspace:** Provided automatically by the module.

**Other routines called directly:** None.

**Other modules used directly:** The GALAHAD package `GALAHAD_SMT` is used by the subroutines `MOP_Ax`, `MOP_getval`, and `MOP_scaleA`.

**Input/output:** Output is provided under the control of the OPTIONAL input arguments `print_level`, `out`, and `error`. The argument `print_level` controls the amount of information printed to the device with unit number `out`; all error messages will be printed to the device with unit number `error`. If the user does not supply any of these optional arguments, then the default values `print_level = 0`, `out = 6`, and `error = 6` are used.

**Restrictions:** `A%n`  $> 0$ , `A%m`  $> 0$ , and `A%type`  $\in \{ 'DENSE', 'COORDINATE', 'SPARSE-BY-ROWS', 'SPARSE-BY-COLUMNS', 'DIAGONAL' \}$ .

**Portability:** ISO Fortran 90.

## 4 EXAMPLE OF USE

Suppose we wish to perform the following operations. We first compute

$$\mathbf{r} \leftarrow \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{r}$$

where

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \quad \alpha = 3, \quad \text{and} \quad \beta = 2.$$

Next we scale the rows of  $\mathbf{A}$  by the vector  $\mathbf{u}$  and columns of  $\mathbf{A}$  by the vector  $\mathbf{v}$ , where

$$\mathbf{u} = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}.$$

In other words, we over-write the matrix  $\mathbf{A}$  with the scaled matrix whose  $(i, j)$ th element is  $u_i a_{i,j} v_j$ . Finally, we retrieve the  $(1, 2)$  element of the scaled matrix.

We may use the following code

```
! THIS VERSION: GALAHAD 4.1 - 2022-11-25 AT 09:00 GMT.
PROGRAM GALAHAD_mop_example
  USE GALAHAD_SMT_double ! double precision version
  USE GALAHAD_MOP_double ! double precision version
  IMPLICIT NONE
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! Define the working precision
  REAL ( KIND = wp ), parameter :: one = 1.0_wp, two = 2.0_wp, three = 3.0_wp
```

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

```

REAL ( KIND = wp ), dimension(:), allocatable :: X, u, v, R
REAL ( KIND = wp ) :: val, alpha, beta
INTEGER :: row, col, out, error, print_level, stat
LOGICAL :: symmetric, transpose
TYPE ( SMT_type ) :: A
! Begin problem data.
A%m = 2 ; A%n = 3 ; A%ne = 6
ALLOCATE( A%row( A%ne ), A%col( A%ne ), A%val( A%ne ), X( A%n ),      &
          u( A%m ), v( A%n ), R( A%m ) )
A%row = (/ 1, 1, 1, 2, 2, 2 /) ; A%col = (/ 1, 2, 3, 1, 2, 3 /)
A%val = (/ 1, 2, 3, 4, 5, 6 /)
CALL SMT_put( A%id, 'Toy 2x3 matrix', stat );
CALL SMT_put( A%type, 'COORDINATE', stat )
X = (/ one, one, one /) ; R = (/ three, three /)
u = (/ two, -one /) ; v = (/ three, one, two /)
! Compute : R <- 3*A X + 2*R
alpha = three ; print_level = 3
beta = two ; symmetric = .false.
out = 6 ; transpose = .false.
error = 6
write(*,*) 'Compute R <- alpha*A*X + beta*R .....'
CALL MOP_Ax( alpha, A, X, beta, R, out, error, print_level, symmetric,      &
             transpose )
! Scale rows of A by u and columns by v.
WRITE( *, * ) 'Scale rows of A by u and columns by v .....'
CALL MOP_scaleA( A, u, v, out, error, print_level, symmetric )
! Get the (1,2) element of scaled matrix.
row = 1 ; col = 2
WRITE( *, * ) 'Obtain the (1,2) element of the scaled matrix A .....'
CALL MOP_getval( A, row, col, val, symmetric, out, error, print_level )
WRITE( *, * ) 'The value of the (1,2) element of the scaled matrix is', val
END PROGRAM GALAHAD_mop_example

```

This produces the following output:

```

Compute R <- alpha*A*X + beta*R .....

*****
*                               *
*          BEGIN: mop_Ax        *
*          GALAHAD sparse matrix operation subroutine          *
*                               *
*****

A%type = COORDINATE
A%id   = Toy 2x3 matrix

transpose = F      m =      2      alpha = 3.0000000000E+00
symmetric = F      n =      3      beta  = 2.0000000000E+00

  A%row      A%col      A%val
  ----      ----      -
    1          1      1.0000000000E+00
    1          2      2.0000000000E+00
    1          3      3.0000000000E+00
    2          1      4.0000000000E+00
    2          2      5.0000000000E+00
    2          3      6.0000000000E+00

```

**All use is subject to the conditions of a BSD-3-Clause License.**  
**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

X	R(in)
-----	-----
1.0000000000E+00	3.0000000000E+00
1.0000000000E+00	3.0000000000E+00
1.0000000000E+00	

R (out)
-----
2.4000000000E+01
5.1000000000E+01

```
*****
*                               END: mop_Ax                               *
*****
```

Scale rows of A by u and columns by v .....

```
*****
*                               BEGIN: mop_scaleA                          *
*   GALAHAD sparse matrix operation subroutine                          *
*****
```

```
-----
Matrix Pre-scaling
-----
```

A%type = COORDINATE  
A%id = Toy 2x3 matrix  
SYMMETRIC = F

(m,n) = ( 2, 3)

A%row	A%col	A%val
-----	-----	-----
1	1	1.0000000000E+00
1	2	2.0000000000E+00
1	3	3.0000000000E+00
2	1	4.0000000000E+00
2	2	5.0000000000E+00
2	3	6.0000000000E+00

u	v
-----	-----
2.0000000000E+00	3.0000000000E+00
-1.0000000000E+00	1.0000000000E+00
	2.0000000000E+00

```
-----
Matrix Post-scaling
-----
```

A%row	A%col	A%val
-----	-----	-----
1	1	6.0000000000E+00
1	2	4.0000000000E+00

---

**All use is subject to the conditions of a BSD-3-Clause License.**  
See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

```

1          3          1.2000000000E+01
2          1          -1.2000000000E+01
2          2          -5.0000000000E+00
2          3          -1.2000000000E+01

```

```

*****
*                               END: mop_scaleA                               *
*****

```

Obtain the (1,2) element of the scaled matrix A .....

```

*****
*                               BEGIN: mop_getval                               *
*      GALAHAD gets a single element of a sparse matrix                        *
*****

```

```

A%type = COORDINATE
A%id   = Toy 2x3 matrix

```

```

m =      2      row =      1      symmetric = F
n =      3      col =      2

```

A%row	A%col	A%val
-----	-----	-----
1	1	6.0000000000E+00
1	2	4.0000000000E+00
1	3	1.2000000000E+01
2	1	-1.2000000000E+01
2	2	-5.0000000000E+00
2	3	-1.2000000000E+01

```

ON EXIT: value = 4.000000000E+00

```

```

*****
*                               END: mop_getval                               *
*****

```

The value of the (1,2) element of the scaled matrix is 4.0000000000000000